Our second algorithm

Let’s look at a simple dataset for motivation:

- **Class**: Play tennis
- **Attributes**:
  - Outlook
  - Temp
  - Humidity
  - Windy

<table>
<thead>
<tr>
<th>Outlook</th>
<th>Temp</th>
<th>Humidity</th>
<th>Windy</th>
<th>Play</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sunny</td>
<td>Hot</td>
<td>High</td>
<td>False</td>
<td>No</td>
</tr>
<tr>
<td>Sunny</td>
<td>Hot</td>
<td>High</td>
<td>True</td>
<td>No</td>
</tr>
<tr>
<td>Overcast</td>
<td>Hot</td>
<td>High</td>
<td>False</td>
<td>Yes</td>
</tr>
<tr>
<td>Rainy</td>
<td>Cool</td>
<td>Normal</td>
<td>False</td>
<td>Yes</td>
</tr>
<tr>
<td>Rainy</td>
<td>Cool</td>
<td>Normal</td>
<td>True</td>
<td>No</td>
</tr>
<tr>
<td>Sunny</td>
<td>Mild</td>
<td>High</td>
<td>False</td>
<td>No</td>
</tr>
<tr>
<td>Sunny</td>
<td>Cool</td>
<td>Normal</td>
<td>False</td>
<td>Yes</td>
</tr>
<tr>
<td>Rainy</td>
<td>Mild</td>
<td>Normal</td>
<td>False</td>
<td>Yes</td>
</tr>
<tr>
<td>Overcast</td>
<td>Mild</td>
<td>High</td>
<td>True</td>
<td>Yes</td>
</tr>
<tr>
<td>Overcast</td>
<td>Hot</td>
<td>Normal</td>
<td>False</td>
<td>Yes</td>
</tr>
<tr>
<td>Rainy</td>
<td>Mild</td>
<td>High</td>
<td>True</td>
<td>No</td>
</tr>
</tbody>
</table>

Decision Trees

- DTs give a different way to identify regions in instance space:
  - Recursively split on values of features to define regions that have single label

- What does this look like with numerical features? (with threshold node tests, for example temp>23)

Decision Trees

- Decision trees can represent any discrete function of discrete attributes!
  - Why?

Decision Trees

- Given training set, can we build a tree that agrees with the data? (yes: easy; why?)

- What is a good decision tree?

- Given training set, how can we build a good tree?
Decision Trees

- Which attribute should we choose for root of tree?
- A numerical example:
  
  \[
  [\text{Pos}, \text{Neg}] \rightarrow [35, 15] \times [15, 35] \\
  [50, 0] + [0, 50] \\
  [10, 30] + [30, 10] + [10, 10] \\
  [25, 25] + [25, 25]
  \]

Assume we picked Outlook for the root. Then we must continue splitting each branch Until ... ?

Final Decision Tree

Decision Trees

- Several selection criteria have been proposed and used.
- Information gain is commonly used (C4.5, J48)
- We need to learn about entropy ...

Entropy \((p_1, \ldots, p_n) = \sum p_i \log \frac{1}{p_i} = -\sum p_i \log p_i \)

For 2 classes \(p_1 = p, p_2 = 1 - p\) and this simplifies to

\[ Entropy(p) = -\sum p \log p - (1 - p) \log (1 - p) \]

Gain(Split) = \(Ent(S) - \sum_j \frac{|S_j|}{|S|} Ent(S_j)\)

Now consider a split \(S \rightarrow S_1, \ldots, S_k\)
where the \(S_i\) are subsets of \(S\)
and may include examples from multiple classes
Example: calculating Gain

- **Outlook = Sunny:**
  
  \[ \text{entropy}(2/5,3/5) = -2/5 \log(2/5) - 3/5 \log(3/5) \approx 0.971 \text{ bits} \]

- **Outlook = Overcast:**
  
  \[ \text{entropy}(1,0) = -1 \log(1) - 0 \log(0) = 0 \text{ bits} \]

- **Outlook = Rainy:**
  
  \[ \text{entropy}(3/5,2/5) = -3/5 \log(3/5) - 2/5 \log(2/5) \approx 0.971 \text{ bits} \]

- **Expected information for attribute:**
  
  \[ \text{info}(3,2,4,0,3,2) = (5/14) \times 0.971 + (4/14) \times 0 + (5/14) \times 0.971 \approx 0.693 \text{ bits} \]

Decision Trees

- **gain(Outlook) =** info([9,5]) – info([2,3],[4,0],[3,2])
  \[ = 0.940 - 0.693 = 0.247 \text{ bits} \]
- **gain(Temperature) =** 0.029 bits
- **gain(Humidity) =** 0.152 bits
- **gain(Windy) =** 0.048 bits

Decision Tree Learning Algorithm

- If data has a pure class
  - Make leaf node with that class
- Otherwise
  - Pick feature to split on
  - Divide data into sub-datasets \( S_j \) according to the feature’s values
  - Recursively build a tree for each subset

Improved Heuristic for Wide Splits

\[ \text{Gain}(\text{Split}) = \text{Ent}(S) - \sum_j \frac{|S_j|}{|S|} \text{Ent}(S_j) \]

Heuristic for wide splits

\[ \text{SplitInfo} = \sum_j \frac{|S_j|}{|S|} \log \frac{|S|}{|S_j|} \]

\[ \text{GainRatio} = \frac{\text{Gain}}{\text{SplitInfo}} \]

Other Criteria / Tasks

The Gini Criterion = \( 4p(1 - p) \)

The [KM] Criterion = \( 2 \sqrt{p(1 - p)} \)

What do these look like?

Criterion for Regression = \( \frac{1}{T} \sum_{t=1}^{T} (v_t - \bar{v})^2 \)

\[ \bar{v} = \frac{1}{T} \sum_{t=1}^{T} v_t \]
Real Valued Attributes

- Naïve treatment makes a very side split with possibly one example per branch.
- Is this good?
- Alternative picks threshold $t$ and tests ($\text{feature} \geq t$) to get a binary split.
- How can we pick $t$?

Missing Attribute Values

- Common in real data
- We can handle this in a way that works across algorithms (that is, also for kNN).
- How?
- But we can do better with a solution tailored for decision trees. How?

The Bad News

- This does not quite work …

Overfitting in DT

- Why Does this happen?
- Few examples at lower levels in tree
- Quantities calculated “not reliable” in this case
- Even worse with “noisy data”
- And when features not sufficiently rich
- Solutions?

Overfitting in DT

- Min # points at leaf for split to be legal
- Stop growing tree if “no information”
- Pruning: grow full tree and then test whether some parts should be removed.
- How? Note that full tree always looks better on training data! so just using accuracy on training data will not work

Overfitting in DT

- Solution 1 (C4.5, J48): uses a confidence interval based on class ratio at leaf and number of examples in the training set.
- This is not fully justified but works well in practice.
- Solution 2: use a validation set. Known as reduced error pruning (REP)
**Pruning in C4.5 / J48**

Standard Normal distribution

\[ (1 - \alpha) = 0.95 \]

\[ \alpha = 0.05 \]

- \( p \) is true error and \( f \) is observed error
- Algorithm uses \( f \sim N\left( p, \frac{p(1-p)}{n} \right) \) and some reasoning to claim that
  
  \[ p \leq f + \sqrt{\frac{f(1-f)}{n}} \cdot Z_\alpha \]

- The error rate at each node is replaced with the upper bound
- Then the best pruning can be chosen

**REP Example**

<table>
<thead>
<tr>
<th>Decision</th>
<th>Errors</th>
</tr>
</thead>
<tbody>
<tr>
<td>Keep</td>
<td>7</td>
</tr>
<tr>
<td>Prune</td>
<td>4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Decision</th>
<th>Errors</th>
</tr>
</thead>
<tbody>
<tr>
<td>Keep</td>
<td>11</td>
</tr>
<tr>
<td>Prune</td>
<td>14</td>
</tr>
</tbody>
</table>

**DT Recap**

- DT divide the example space through recursive splits of feature values
- Recursive learning algorithm relies of good choice of root attribute
- IG and other criteria are used for choice
- Several variants, improvements and generalizations
- Overfitting is a significant issue: solved by pruning or other methods